CS 595: Assignment #8

Due on Friday, November 14, 2014

Dr Nelson 4:20pm

Victor Nwala

Contents

Problem 1

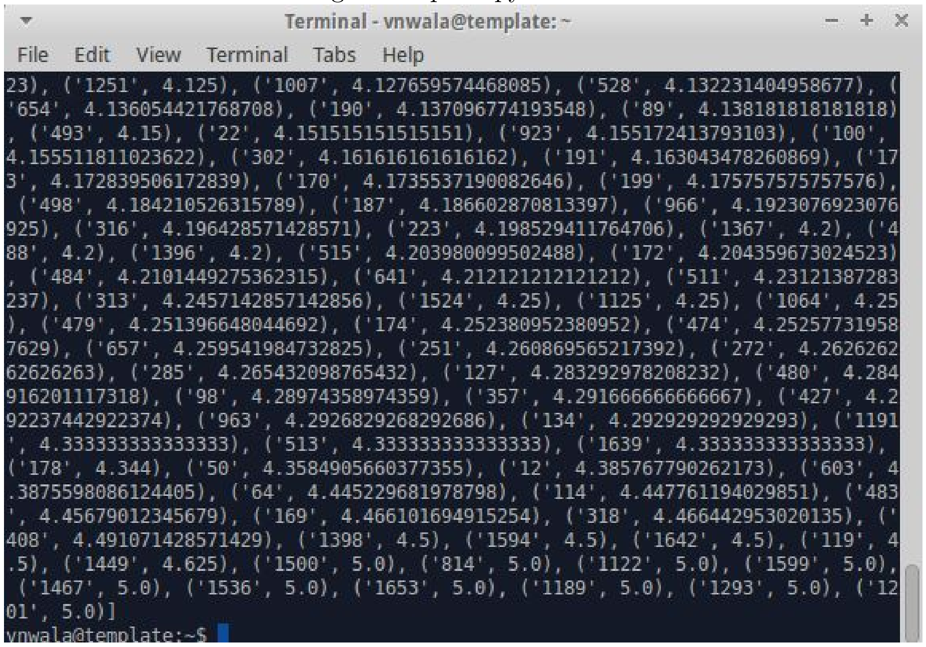
1. What 5 movies have the highest average ratings? Show the movies and their ratings sorted by their average ratings.
The code sorts the result from least to highest rated movie. Their rankings are:
1) 1201—Marlene Dietrich: Shadow and Light (1996) 2) 1293—Star Kid (1997) 3) 1189—Prefontaine (1997) 4) 1653—Entertaining Angels: The Dorothy Day Story (1996) 5) 1536—Aiqing wansui (1994) All with average ratings of 5.0

Listing 1: Python script solving problem 1

```
from collections import Counter
from collections import defaultdict
from operator import itemgetter, attrgetter
from collections import OrderedDict

path='/home/vnwala/ml-100k'
def getMovieData():
    movie = {}
    for line in open(path + '/u.data'):
        (user, movieid, rating, ts) = line.split('\t')

        if( movieid in movie ):
            movie[movieid].append(float(rating))
        else:
            movie[movieid] = []
            movie[movieid].append(float(rating))

    return movie

movie = getMovieData()

sums = {}

for movieid in movie:
    a = sum(movie[movieid])/len(movie[movieid])
    if( movieid in movie ):
        sums[movieid] = a
    else:
        sums[movieid] = a

sums = sorted(sums.items(), key=lambda x: x[1])
print sums
```

Problem 1 continued on next page. . .

Figure 1: ques1.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
23), ('1251', 4.125), ('1007', 4.127659574468085), ('528', 4.132231404958677), (
'654', 4.136054421768708), ('190', 4.137096774193548), ('89', 4.138181818181818)
, ('493', 4.15), ('22', 4.151515151515151), ('923', 4.155172413793103), ('100',
4.155511811023622), ('302', 4.161616161616162), ('191', 4.163043478260869), ('17
3', 4.172839506172839), ('170', 4.1735537190082646), ('199', 4.175757575757576),
 ('498', 4.184210526315789), ('187', 4.186602870813397), ('966', 4.1923076923076
925), ('316', 4.196428571428571), ('223', 4.198529411764706), ('1367', 4.2), ('4
88', 4.2), ('1396', 4.2), ('515', 4.203980099502488), ('172', 4.204359673024523)
, ('484', 4.2101449275362315), ('641', 4.212121212121212), ('511', 4.2312138728
237), ('313', 4.2457142857142856), ('1524', 4.25), ('1125', 4.25), ('1064', 4.25
), ('479', 4.251396648044692), ('174', 4.252380952380952), ('474', 4.2525773195
87629), ('657', 4.259541984732825), ('251', 4.260869565217392), ('272', 4.2626262
62626263), ('285', 4.265432098765432), ('127', 4.283292978208232), ('480', 4.284
916201117318), ('98', 4.28974358974359), ('357', 4.291666666666667), ('427', 4.2
92237442922374), ('963', 4.2926829268292686), ('134', 4.292929292929293), ('1191
', 4.333333333333333), ('513', 4.333333333333333), ('1639', 4.333333333333333),
('178', 4.344), ('50', 4.3584905660377355), ('12', 4.385767790262173), ('603', 4
.3875598086124405), ('64', 4.445229681978798), ('114', 4.447761194029851), ('483
', 4.45679012345679), ('169', 4.466101694915254), ('318', 4.466442953020135), ('
408', 4.491071428571429), ('1398', 4.5), ('1594', 4.5), ('1642', 4.5), ('119', 4
.5), ('1449', 4.625), ('1500', 5.0), ('814', 5.0), ('1122', 5.0), ('1599', 5.0),
 ('1467', 5.0), ('1536', 5.0), ('1653', 5.0), ('1189', 5.0), ('1293', 5.0), ('12
01', 5.0)]
vnwala@template:~$
```

Problem 2

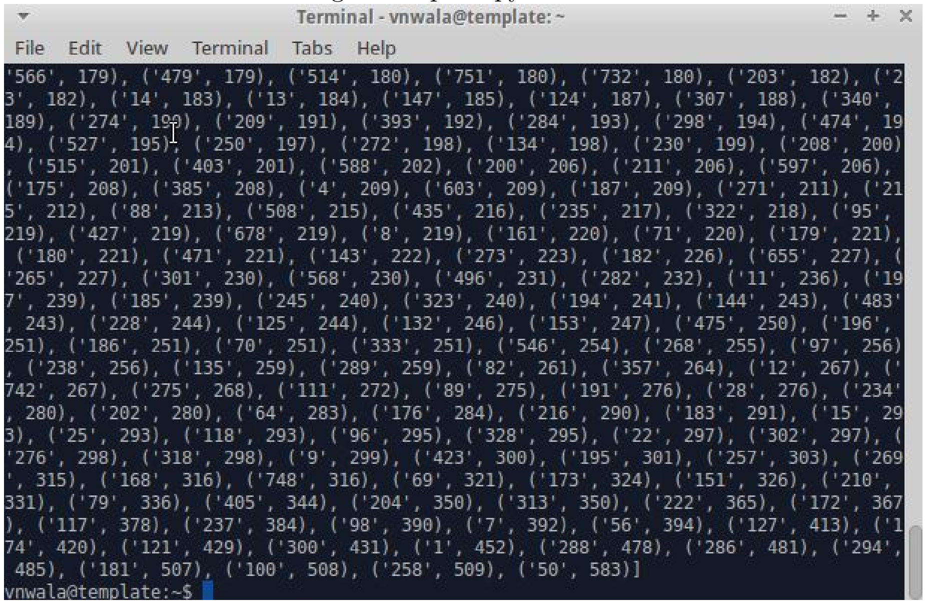
2. What 5 movies received the most ratings? Show the movies and the number of ratings sorted by number of ratings.
The code sorts the result from least to most rated movie. Their rankings are:
1) 50—Star Wars (1977) with 583 ratings 2) 258—Contact (1997) with 509 ratings 3) 100—Fargo (1996) with 508 ratings 4) 181—Return of the Jedi (1983) with 507 ratings 5) 294—Liar Liar (1997) with 485 ratings

Listing 2: Python script solving problem 2

```
from collections import Counter
from collections import defaultdict
from operator import itemgetter, attrgetter
from collections import OrderedDict

path='/home/vnwala/ml-100k'
def getMovieData():
     movie = {}
     for line in open(path + '/u.data'):
          (user, movieid, rating, ts) = line.split('\t')

          if( movieid in movie ):
               movie[movieid].append(float(rating))
          else:
               movie[movieid] = []
               movie[movieid].append(float(rating))

     return movie

movie = getMovieData()

sums2 = {}

for movieid in movie:
     a = len(movie[movieid])
     if( movieid in movie ):
          sums2[movieid] = a
     else:
          sums2[movieid] = a
sums2 = sorted(sums2.items(), key=lambda x: x[1])
print sums2
```

Problem 2 continued on next page. . .

Figure 2: ques2.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
'566', 179), ('479', 179), ('514', 180), ('751', 180), ('732', 180), ('203', 182), ('2
3', 182), ('14', 183), ('13', 184), ('147', 185), ('124', 187), ('307', 188), ('340',
189), ('274', 190), ('209', 191), ('393', 192), ('284', 193), ('298', 194), ('474', 19
4), ('527', 195), ('250', 197), ('272', 198), ('134', 198), ('230', 199), ('208', 200)
, ('515', 201), ('403', 201), ('588', 202), ('200', 206), ('211', 206), ('597', 206),
('175', 208), ('385', 208), ('4', 209), ('603', 209), ('187', 209), ('271', 211), ('21
5', 212), ('88', 213), ('508', 215), ('435', 216), ('235', 217), ('322', 218), ('95',
219), ('427', 219), ('678', 219), ('8', 219), ('161', 220), ('71', 220), ('179', 221),
 ('180', 221), ('471', 221), ('143', 222), ('273', 223), ('182', 226), ('655', 227), (
'265', 227), ('301', 230), ('568', 230), ('496', 231), ('282', 232), ('11', 236), ('19
7', 239), ('185', 239), ('245', 240), ('323', 240), ('194', 241), ('144', 243), ('483'
, 243), ('228', 244), ('125', 244), ('132', 246), ('153', 247), ('475', 250), ('196',
251), ('186', 251), ('70', 251), ('333', 251), ('546', 254), ('268', 255), ('97', 256)
, ('238', 256), ('135', 259), ('289', 259), ('82', 261), ('357', 264), ('12', 267), ('
742', 267), ('275', 268), ('111', 272), ('89', 275), ('191', 276), ('28', 276), ('234'
, 280), ('202', 280), ('64', 283), ('176', 284), ('216', 290), ('183', 291), ('15', 29
3), ('25', 293), ('118', 293), ('96', 295), ('328', 295), ('22', 297), ('302', 297), (
'276', 298), ('318', 298), ('9', 299), ('423', 300), ('195', 301), ('257', 303), ('269
', 315), ('168', 316), ('748', 316), ('69', 321), ('173', 324), ('151', 326), ('210',
331), ('79', 336), ('405', 344), ('204', 350), ('313', 350), ('222', 365), ('172', 367
), ('117', 378), ('237', 384), ('98', 390), ('7', 392), ('56', 394), ('127', 413), ('1
74', 420), ('121', 429), ('300', 431), ('1', 452), ('288', 478), ('286', 481), ('294',
 485), ('181', 507), ('100', 508), ('258', 509), ('50', 583)]
vnwala@template:~$
```

Problem 3

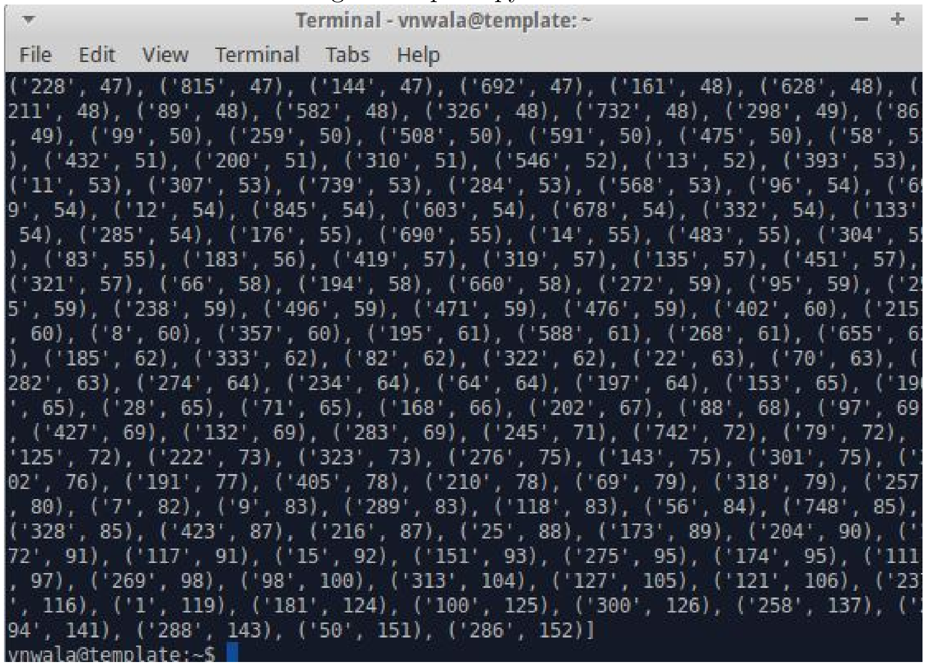
3. What 5 movies were rated the highest on average by women? Show the movies and their ratings sorted by ratings.
1) 286—English Patient, The (1996) with 152 female raters 2) 50—Star Wars (1977) with 151 female raters 3) 288—Scream (1996) with 143 female raters 4) 294—Liar Liar (1997) with 141 female raters 5) 258—Contact (1997) with 137 female raters

Listing 3: Python script solving problem 3

```python
path='/home/vnwala/ml-100k'
user1 = {}
movie = {}
female_raters ={}

for line in open(path + '/u.user'):
     (userid , age, gender, occupation, zipcode) = line.split('|')
     if( gender ==  'F' ):
          user1[userid] = (age, gender, occupation, zipcode)



          for line in open(path + '/u.data'):
               (user, movieid, rating, ts) = line.split('\t')
               if (user == userid):
                    if( movieid in movie ):
                         movie[movieid].append(userid)
                    else:
                         movie[movieid] = []
                         movie[movieid].append(userid)
for movieid in movie:
     female_raters[movieid] = len(movie[movieid])

female_raters = sorted(female_raters.items(), key=lambda x: x[1])

print female_raters
```

Problem 3 continued on next page...

Figure 3: ques3.py at work



```
('228', 47), ('815', 47), ('144', 47), ('692', 47), ('161', 48), ('628', 48), (
211', 48), ('89', 48), ('582', 48), ('326', 48), ('732', 48), ('298', 49), ('86
, 49), ('99', 50), ('259', 50), ('508', 50), ('591', 50), ('475', 50), ('58', 5
), ('432', 51), ('200', 51), ('310', 51), ('546', 52), ('13', 52), ('393', 53),
('11', 53), ('307', 53), ('739', 53), ('284', 53), ('568', 53), ('96', 54), ('6
9', 54), ('12', 54), ('845', 54), ('603', 54), ('678', 54), ('332', 54), ('133'
 54), ('285', 54), ('176', 55), ('690', 55), ('14', 55), ('483', 55), ('304', 5
), ('83', 55), ('183', 56), ('419', 57), ('319', 57), ('135', 57), ('451', 57),
('321', 57), ('66', 58), ('194', 58), ('660', 58), ('272', 59), ('95', 59), ('2
5', 59), ('238', 59), ('496', 59), ('471', 59), ('476', 59), ('402', 60), ('215
, 60), ('8', 60), ('357', 60), ('195', 61), ('588', 61), ('268', 61), ('655', 6
), ('185', 62), ('333', 62), ('82', 62), ('322', 62), ('22', 63), ('70', 63), (
282', 63), ('274', 64), ('234', 64), ('64', 64), ('197', 64), ('153', 65), ('19
', 65), ('28', 65), ('71', 65), ('168', 66), ('202', 67), ('88', 68), ('97', 69
, ('427', 69), ('132', 69), ('283', 69), ('245', 71), ('742', 72), ('79', 72),
'125', 72), ('222', 73), ('323', 73), ('276', 75), ('143', 75), ('301', 75), ('
02', 76), ('191', 77), ('405', 78), ('210', 78), ('69', 79), ('318', 79), ('257
, 80), ('7', 82), ('9', 83), ('289', 83), ('118', 83), ('56', 84), ('748', 85),
('328', 85), ('423', 87), ('216', 87), ('25', 88), ('173', 89), ('204', 90), ('
72', 91), ('117', 91), ('15', 92), ('151', 93), ('275', 95), ('174', 95), ('111
, 97), ('269', 98), ('98', 100), ('313', 104), ('127', 105), ('121', 106), ('23
', 116), ('1', 119), ('181', 124), ('100', 125), ('300', 126), ('258', 137), ('
94', 141), ('288', 143), ('50', 151), ('286', 152)]
vnwala@template:~$
```

Problem 4

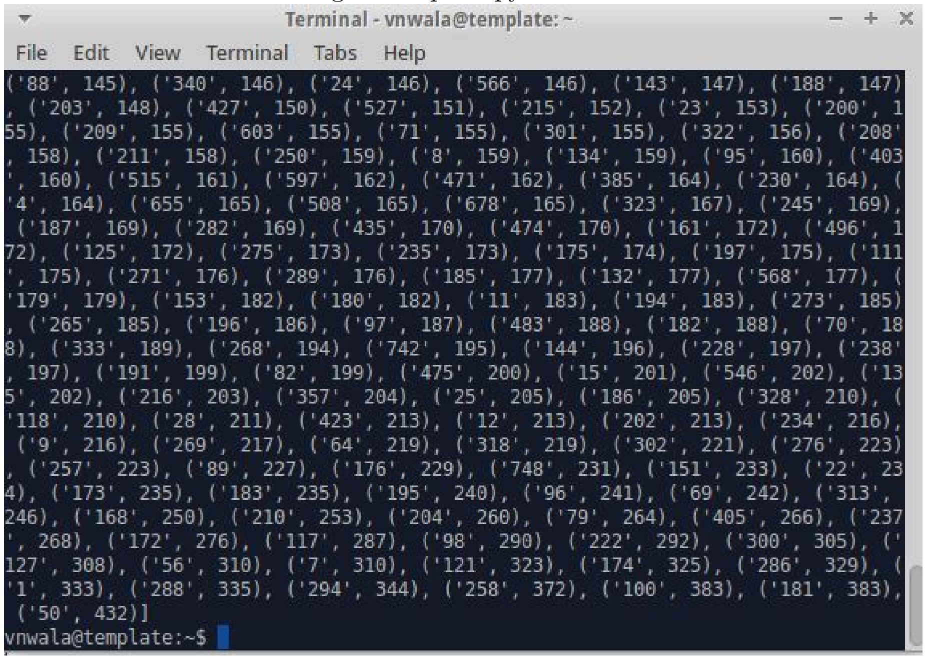
4. What 5 movies were rated the highest on average by men? Show the movies and their ratings sorted by ratings.
1) 50—Star Wars (1977) with 432 male raters 2) 181—Return of the Jedi (1983) with 383 male raters 3) 100—Fargo (1996) with 383 male raters 4) 258—Contact (1997) with 372 male raters 5) 294—Liar Liar (1997) with 344 male raters

Listing 4: Python script solving problem 4

```
path='/home/vnwala/ml-100k'
user1 = {}
movie = {}
male_raters ={}

for line in open(path + '/u.user'):
    (userid , age, gender, occupation, zipcode) = line.split('|')
    if( gender ==  'M' ):
        user1[userid] = (age, gender, occupation, zipcode)



        for line in open(path + '/u.data'):
            (user, movieid, rating, ts) = line.split('\t')
            if (user == userid):
                if( movieid in movie ):
                    movie[movieid].append(userid)
                else:
                    movie[movieid] = []
                    movie[movieid].append(userid)
for movieid in movie:
    male_raters[movieid] = len(movie[movieid])

male_raters = sorted(male_raters.items(), key=lambda x: x[1])

print male_raters
```

Problem 4 continued on next page. . .

Figure 4: ques4.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
('88', 145), ('340', 146), ('24', 146), ('566', 146), ('143', 147), ('188', 147)
, ('203', 148), ('427', 150), ('527', 151), ('215', 152), ('23', 153), ('200', 1
55), ('209', 155), ('603', 155), ('71', 155), ('301', 155), ('322', 156), ('208'
, 158), ('211', 158), ('250', 159), ('8', 159), ('134', 159), ('95', 160), ('403
', 160), ('515', 161), ('597', 162), ('471', 162), ('385', 164), ('230', 164), (
'4', 164), ('655', 165), ('508', 165), ('678', 165), ('323', 167), ('245', 169),
 ('187', 169), ('282', 169), ('435', 170), ('474', 170), ('161', 172), ('496', 1
72), ('125', 172), ('275', 173), ('235', 173), ('175', 174), ('197', 175), ('111
', 175), ('271', 176), ('289', 176), ('185', 177), ('132', 177), ('568', 177), (
'179', 179), ('153', 182), ('180', 182), ('11', 183), ('194', 183), ('273', 185)
, ('265', 185), ('196', 186), ('97', 187), ('483', 188), ('182', 188), ('70', 18
8), ('333', 189), ('268', 194), ('742', 195), ('144', 196), ('228', 197), ('238'
, 197), ('191', 199), ('82', 199), ('475', 200), ('15', 201), ('546', 202), ('13
5', 202), ('216', 203), ('357', 204), ('25', 205), ('186', 205), ('328', 210), (
'118', 210), ('28', 211), ('423', 213), ('12', 213), ('202', 213), ('234', 216),
 ('9', 216), ('269', 217), ('64', 219), ('318', 219), ('302', 221), ('276', 223)
, ('257', 223), ('89', 227), ('176', 229), ('748', 231), ('151', 233), ('22', 23
4), ('173', 235), ('183', 235), ('195', 240), ('96', 241), ('69', 242), ('313',
246), ('168', 250), ('210', 253), ('204', 260), ('79', 264), ('405', 266), ('237
', 268), ('172', 276), ('117', 287), ('98', 290), ('222', 292), ('300', 305), ('
127', 308), ('56', 310), ('7', 310), ('121', 323), ('174', 325), ('286', 329), (
'1', 333), ('288', 335), ('294', 344), ('258', 372), ('100', 383), ('181', 383),
 ('50', 432)]
vnwala@template:~$
```

Problem 5

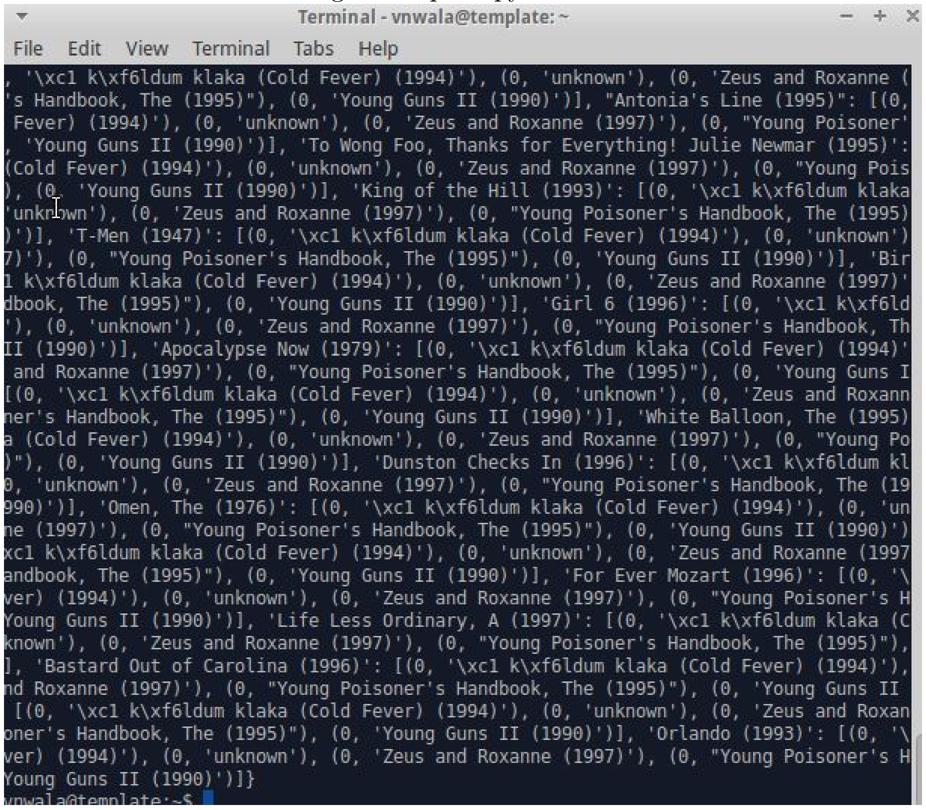
5. What movie received ratings most like Top Gun? Which movie received ratings that were least like Top Gun (negative correlation)?
Some movies rated the most like like Top Gun are:
1) Mr. Smith Goes to Washington (1939) 2) (Cold Fever) (1994) 3) Young Guns II (1990) 4) Young Poisoner's Handbook, The (1995) 6) Zeus and Roxanne (1997) 7) Young Poisoner's Handbook, The (1995) 8) Out to Sea (1997) 9) Old Yeller (1957) 10) Hungarian Fairy Tale, A (1987)

Listing 5: Python script solving problem 5

```python
from collections import Counter
from collections import defaultdict
from operator import itemgetter, attrgetter
from collections import OrderedDict

path='/home/vnwala/ml-100k'





path='/home/vnwala/ml-100k'


def sim_distance(prefs, p1, p2):
     '''
     Returns a distance-based similarity score for person1 and person2.
     '''
     # Get the list of shared_items
     si = {}
     for item in prefs[p1]:
          if item in prefs[p2]:
               si[item] = 1
     # If they have no ratings in common, return 0
     if len(si) == 0:
          return 0
     # Add up the squares of all the differences
     sum_of_squares = sum([pow(prefs[p1][item] - prefs[p2][item], 2) for item in
     prefs[p1] if item in prefs[p2]])
     return 1 / (1 + sum_of_squares)


def sim_pearson(prefs, p1, p2):
     '''
     Returns the Pearson correlation coefficient for p1 and p2.
     '''
     # Get the list of mutually rated items
     si = {}
     for item in prefs[p1]:
          if item in prefs[p2]:
               si[item] = 1
     # If they are no ratings in common, return 0
     if len(si) == 0:
          return 0
```

```
    # Sum calculations
    n = len(si)
    # Sums of all the preferences
    sum1 = sum([prefs[p1][it] for it in si])
    sum2 = sum([prefs[p2][it] for it in si])
    # Sums of the squares
    sum1Sq = sum([pow(prefs[p1][it], 2) for it in si])
    sum2Sq = sum([pow(prefs[p2][it], 2) for it in si])
    # Sum of the products
    pSum = sum([prefs[p1][it] * prefs[p2][it] for it in si])
    # Calculate r (Pearson score)
    num = pSum - sum1 * sum2 / n
    den = sqrt((sum1Sq - pow(sum1, 2) / n) * (sum2Sq - pow(sum2, 2) / n))
    if den == 0:
        return 0
    r = num / den
    return r


def topMatches(
prefs,
person,
n=5,
similarity=sim_pearson,
):
    '''
    Returns the best matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
    scores = [(similarity(prefs, person, other), other) for other in prefs
    if other != person]
    scores.sort()
    scores.reverse()
    return scores[0:n]






def transformPrefs(prefs):
    '''
    Transform the recommendations into a mapping where persons are described
    with interest scores for a given title e.g. {title: person} instead of
    {person: title}.
    '''
    result = {}
    for person in prefs:
        for item in prefs[person]:
            result.setdefault(item, {})
    # Flip item and person
        result[item][person] = prefs[person][item]
    return result







def calculateSimilarItems(prefs, n=5):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''
    result = {}
    # Invert the preference matrix to be item-centric
    itemPrefs = transformPrefs(prefs)
    c = 0
    for item in itemPrefs:
    # Status updates for large datasets

        scores = topMatches(itemPrefs, 'Top Gun (1986)', n=n, similarity=
            sim_distance)
        result[item] = scores
    return result









# Get movie titles
movies = {}
for line in open(path + '/u.item'):
    (id, title) = line.split('|')[0:2]
    movies[id] = title
# Load data
prefs = {}
for line in open(path + '/u.data'):
    (user, movieid, rating, ts) = line.split('\t')
    prefs.setdefault(user, {})
    prefs[user][movies[movieid]] = float(rating)


output = calculateSimilarItems(prefs)

print output
```

Figure 5: ques5.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (
's Handbook, The (1995)"), (0, 'Young Guns II (1990)')], "Antonia's Line (1995)": [(0,
 Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner'
, 'Young Guns II (1990)')], 'To Wong Foo, Thanks for Everything! Julie Newmar (1995)':
(Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Pois
), (0, 'Young Guns II (1990)')], 'King of the Hill (1993)': [(0, '\xc1 k\xf6ldum klaka
'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner's Handbook, The (1995)
)')], 'T-Men (1947)': [(0, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'unknown')
7)'), (0, "Young Poisoner's Handbook, The (1995)"), (0, 'Young Guns II (1990)')], 'Bir
1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'
dbook, The (1995)"), (0, 'Young Guns II (1990)')], 'Girl 6 (1996)': [(0, '\xc1 k\xf6ld
'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner's Handbook, Th
II (1990)')], 'Apocalypse Now (1979)': [(0, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'
 and Roxanne (1997)'), (0, "Young Poisoner's Handbook, The (1995)"), (0, 'Young Guns I
[(0, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxann
ner's Handbook, The (1995)"), (0, 'Young Guns II (1990)')], 'White Balloon, The (1995)
a (Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Po
)"), (0, 'Young Guns II (1990)')], 'Dunston Checks In (1996)': [(0, '\xc1 k\xf6ldum kl
0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner's Handbook, The (19
990)')], 'Omen, The (1976)': [(0, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'un
ne (1997)'), (0, "Young Poisoner's Handbook, The (1995)"), (0, 'Young Guns II (1990)')
xc1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997
andbook, The (1995)"), (0, 'Young Guns II (1990)')], 'For Ever Mozart (1996)': [(0, '\
ver) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner's H
Young Guns II (1990)')], 'Life Less Ordinary, A (1997)': [(0, '\xc1 k\xf6ldum klaka (C
known'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner's Handbook, The (1995)"),
], 'Bastard Out of Carolina (1996)': [(0, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'),
nd Roxanne (1997)'), (0, "Young Poisoner's Handbook, The (1995)"), (0, 'Young Guns II
 [(0, '\xc1 k\xf6ldum klaka (Cold Fever) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxan
oner's Handbook, The (1995)"), (0, 'Young Guns II (1990)')], 'Orlando (1993)': [(0, '\
ver) (1994)'), (0, 'unknown'), (0, 'Zeus and Roxanne (1997)'), (0, "Young Poisoner's H
Young Guns II (1990)')]}
vnwala@template:~$
```

Problem 6

6. Which 5 raters rated the most films? Show the raters' IDs and the number of films each rated.
The solution is computed in this format, rater id:number of movies rated
1) 405:737 2) 655:685 3) 13:636 4) 450:540 5) 276:518

Listing 6: Python script solving problem 6

```python
from math import sqrt
import os, sys


def loadMovieLens(path='/home/vnwala/ml-100k/'):
    movies = {}
    for line in open(path + 'u.item'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
    prefs = {}
    for line in open(path + 'u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movieid] = float(rating)
    return prefs, movies


def aggregateMovieAndUserData(path='/home/vnwala/ml-100k/'):

    try:

        movies = {}
        aggregateMovieData = []
        for line in open(path + 'u.item'):
            (id, title) = line.split('|')[0:2]
            movies[id] = (title, [], -1)

        users = {}

        for line in open(path + 'u.data'):
            (user, movieid, rating, ts) = line.split('\t')

            user = user.strip()
            movieid = movieid.strip()
            rating = rating.strip()
            ts = ts.strip()

            users.setdefault(user, {})
            users[user][movieid] = float(rating)

            movies[movieid][1].append(float(rating))

        for movieId, tupleData in movies.items():
            averageRating = sum(tupleData[1]) / float(len(tupleData[1]))

```

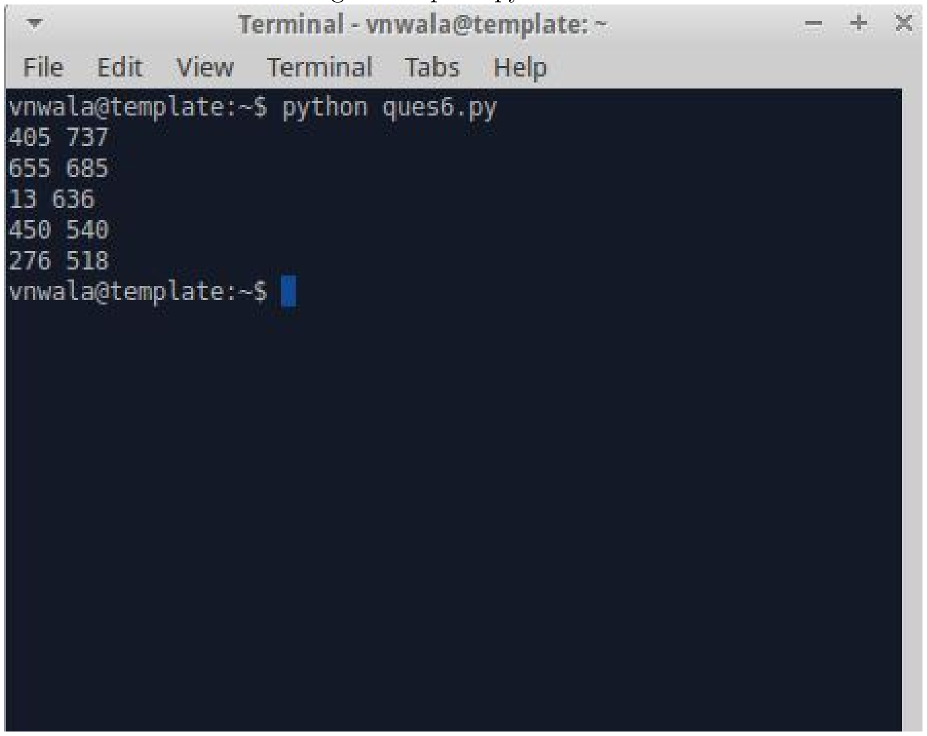
Problem 6 continued on next page. . .

```
                movietuples = (movieId, tupleData[0], averageRating, len(tupleData[1])
                    )
                aggregateMovieData.append(movietuples)


            aggregateUserData = []
            for line in open(path + 'u.user'):
                (userId, age, gender, occupation, zipCode) = line.split('|')
                userTuples = (userId, gender, age, users[userId])
                aggregateUserData.append(userTuples)


        except:
            exc_type, exc_obj, exc_tb = sys.exc_info()
            fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
            print(fname, exc_tb.tb_lineno, sys.exc_info() )
            return



        return aggregateMovieData, aggregateUserData, movies


def getHighestRatedMovies(movies, count):

    if( count > 0 and count < len(movies) ):
        movies = sorted(movies, key=lambda tup: tup[2], reverse=True)
        i = 1
        for movie in movies:
            print movie

            if( i == count ):
                break
            i = i + 1


def getFemaleAndMaleData(aggregateUsers):
    aggregateUsersFemale = []
    aggregateUsersMale = []
    if( len(aggregateUsers) > 0 ):

        for user in aggregateUsers:
            if( user[1] == 'F' ):
                aggregateUsersFemale.append(user)
            else:
                aggregateUsersMale.append(user)


    return aggregateUsersFemale, aggregateUsersMale



def getHighestMovieRaters(aggregateUsers, count):
```

Problem 6 continued on next page. . .

```
    if( len(aggregateUsers) > 0 and count > 0 ):
        raterIDRatedMoviesCount = []

        for rater in aggregateUsers:

            #raterTuple: rater id, number of movies rated
            raterTuple = (rater[0], len(rater[3]))
            raterIDRatedMoviesCount.append(raterTuple)

        raterIDRatedMoviesCount = sorted(raterIDRatedMoviesCount, key=lambda tup:
            tup[1], reverse=True)

        i = 1
        for rater in raterIDRatedMoviesCount:
            print rater[0], rater[1]

            if( i == count ):
                break
            i = i + 1


aggregateMovies, aggregateUsers, movies = aggregateMovieAndUserData()

aggregateUsersFemale, aggregateUsersMale = getFemaleAndMaleData(aggregateUsers)

getHighestMovieRaters(aggregateUsers, 5)
```

Figure 6: ques6.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
vnwala@template:~$ python ques6.py
405 737
655 685
13 636
450 540
276 518
vnwala@template:~$
```

Problem 7

7. Which 5 raters most agreed with each other? Show the raters' IDs and Pearson's r, sorted by r.
Some 5 raters that mostly agree with each other, represented by the IDs are: '100', '105', '107', '111' and '112'

Listing 7: Python script solving problem 7

```
from math import sqrt
import os, sys




def loadMovieLens(path='/home/vnwala/ml-100k/'):
  # Get movie titles
    movies = {}
    for line in open(path + 'u.item'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open(path + 'u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        #prefs[user][movies[movieid]] = float(rating)
        prefs[user][movieid] = float(rating)
    return prefs, movies

prefs, movies = loadMovieLens()



def sim_pearson(prefs,p1,p2):
    # Get the list of mutually rated shared_items
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1

    # Find the number of elements
    n=len(si)

    # if they have no ratings in common, return 0
    if n==0: return 0

    # Add up all the preferences
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])

    # Sum up the squares
    sum1Sq=sum([pow(prefs[p1][it],2) for it in si])
    sum2Sq=sum([pow(prefs[p2][it],2) for it in si])

    # Sum up the products
```

Problem 7 continued on next page. . .

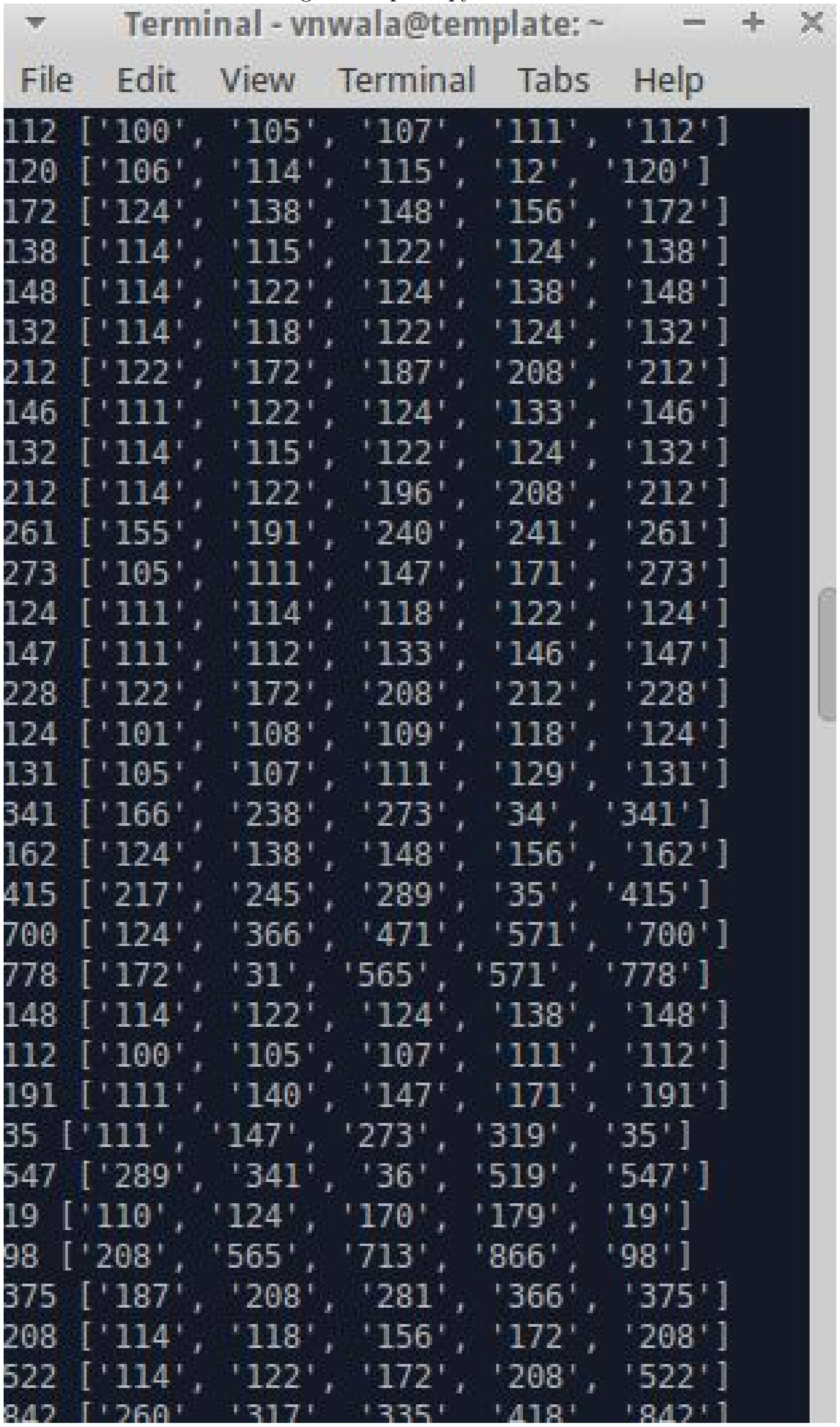
```
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])

    # Calculate Pearson score
    num=pSum-(sum1*sum2/n)
    den=sqrt((sum1Sq-pow(sum1,2)/n)*(sum2Sq-pow(sum2,2)/n))
    if den==0: return 0

    r=num/den

    return r










def topMatches(prefs,person,n=5,similarity=sim_pearson, reverseSimilarityFlag=True
    ):
    scores=[(similarity(prefs,person,other),other) for other in prefs if other!=
        person]

    # Sort the list so the highest scores appear at the top
    scores.sort()

    if( reverseSimilarityFlag ):
        scores.reverse()

    return scores[0:n]






def calculateSimilarItems(prefs, similarityMetric, n=10, reverseSimilarityFlag=
    True, transformMatrixFlag=True):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''

    result = {}
    # Invert the preference matrix to be item-centric

    if( transformMatrixFlag ):
        #with transform: movie top similarity
        itemPrefs = transformPrefs(prefs)
    else:
        #without transform: user top similarity
        itemPrefs = prefs
    #c = 0
    for item in itemPrefs:

        scores = topMatches(itemPrefs, item, n=n, similarity=similarityMetric,
            reverseSimilarityFlag=reverseSimilarityFlag)

        result[item] = scores
    return result


def sim_distance(prefs,person1,person2):
    # Get the list of shared_items
    si = {}
    for item in prefs[person1]:
        if item in prefs[person2]:
            si[item]=1

    # if they have no ratins in common, return 0
    if len(si)==0: return 0

    # Add up the squares of all the differences
    sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2) for item
        in si])

    return 1/(1+sqrt(sum_of_squares))






userSimilarityMatrix = calculateSimilarItems(prefs=prefs, similarityMetric=
    sim_distance, n=5, reverseSimilarityFlag=False, transformMatrixFlag=False)
userSimilarityArrayOfTuples = []
for userId, userAttr in userSimilarityMatrix.items():

    totalSimilarity = 0
    similarUsersArray = []
    for scoreAnduserId in userAttr:
        score = scoreAnduserId[0]
        userId = scoreAnduserId[1]

        similarUsersArray.append(userId)
        totalSimilarity = totalSimilarity + score

    userTupleData = (userId, similarUsersArray, totalSimilarity )
    userSimilarityArrayOfTuples.append(userTupleData)


for userTuple in userSimilarityArrayOfTuples:

    userId = userTuple[0]
```

Problem 7 continued on next page. . .

```
    userSimilarItems = userTuple[1]
    totalSim = userTuple[2]


    #any one of these qualifies
    if( totalSim == 0 ):
        print userId, userSimilarItems
```

Problem 7 continued on next page. . .

Figure 7: ques7.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
112 ['100', '105', '107', '111', '112']
120 ['106', '114', '115', '12', '120']
172 ['124', '138', '148', '156', '172']
138 ['114', '115', '122', '124', '138']
148 ['114', '122', '124', '138', '148']
132 ['114', '118', '122', '124', '132']
212 ['122', '172', '187', '208', '212']
146 ['111', '122', '124', '133', '146']
132 ['114', '115', '122', '124', '132']
212 ['114', '122', '196', '208', '212']
261 ['155', '191', '240', '241', '261']
273 ['105', '111', '147', '171', '273']
124 ['111', '114', '118', '122', '124']
147 ['111', '112', '133', '146', '147']
228 ['122', '172', '208', '212', '228']
124 ['101', '108', '109', '118', '124']
131 ['105', '107', '111', '129', '131']
341 ['166', '238', '273', '34', '341']
162 ['124', '138', '148', '156', '162']
415 ['217', '245', '289', '35', '415']
700 ['124', '366', '471', '571', '700']
778 ['172', '31', '565', '571', '778']
148 ['114', '122', '124', '138', '148']
112 ['100', '105', '107', '111', '112']
191 ['111', '140', '147', '171', '191']
35 ['111', '147', '273', '319', '35']
547 ['289', '341', '36', '519', '547']
19 ['110', '124', '170', '179', '19']
98 ['208', '565', '713', '866', '98']
375 ['187', '208', '281', '366', '375']
208 ['114', '118', '156', '172', '208']
522 ['114', '122', '172', '208', '522']
842 ['260', '317', '335', '418', '842']
```

Problem 8

8. Which 5 raters most disagreed with each other (negative correlation)? Show the raters' IDs and Pearson's r, sorted by r.

Listing 8: Python script solving problem 8

```
from math import sqrt
import os, sys




def loadMovieLens(path='/home/vnwala/ml-100k/'):
  # Get movie titles
    movies = {}
    for line in open(path + 'u.item'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
  # Load data
    prefs = {}
    for line in open(path + 'u.data'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        #prefs[user][movies[movieid]] = float(rating)
        prefs[user][movieid] = float(rating)
    return prefs, movies

prefs, movies = loadMovieLens()



def sim_pearson(prefs,p1,p2):
    # Get the list of mutually rated shared_items
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1

    # Find the number of elements
    n=len(si)

    # if they have no ratings in common, return 0
    if n==0: return 0

    # Add up all the preferences
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])

    # Sum up the squares
    sum1Sq=sum([pow(prefs[p1][it],2) for it in si])
    sum2Sq=sum([pow(prefs[p2][it],2) for it in si])

    # Sum up the products
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])
```

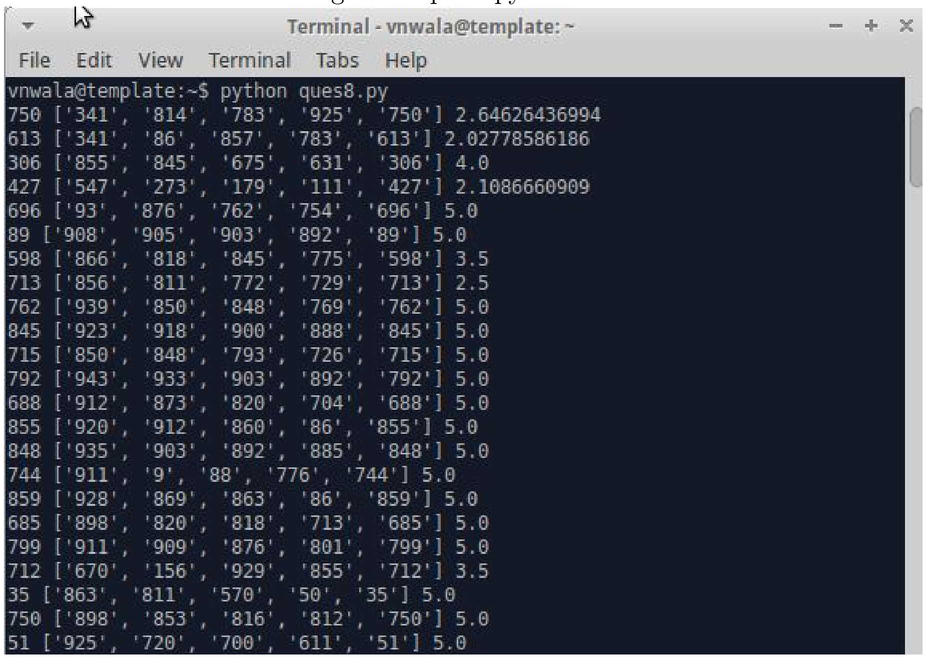
Problem 8 continued on next page. . .

```
    # Calculate Pearson score
    num=pSum-(sum1*sum2/n)
    den=sqrt((sum1Sq-pow(sum1,2)/n)*(sum2Sq-pow(sum2,2)/n))
    if den==0: return 0

    r=num/den

    return r









def topMatches(prefs,person,n=5,similarity=sim_pearson, reverseSimilarityFlag=True
    ):
    scores=[(similarity(prefs,person,other),other) for other in prefs if other!=
        person]

    # Sort the list so the highest scores appear at the top
    scores.sort()

    if( reverseSimilarityFlag ):
        scores.reverse()

    return scores[0:n]






def calculateSimilarItems(prefs, similarityMetric, n=10, reverseSimilarityFlag=
    True, transformMatrixFlag=True):
    '''
    Create a dictionary of items showing which other items they are
    most similar to.
    '''

    result = {}
    # Invert the preference matrix to be item-centric

    if( transformMatrixFlag ):
        #with transform: movie top similarity
        itemPrefs = transformPrefs(prefs)
    else:
        #without transform: user top similarity
        itemPrefs = prefs
    #c = 0
    for item in itemPrefs:

        scores = topMatches(itemPrefs, item, n=n, similarity=similarityMetric,
            reverseSimilarityFlag=reverseSimilarityFlag)

        result[item] = scores
    return result


def sim_distance(prefs,person1,person2):
    # Get the list of shared_items
    si = {}
    for item in prefs[person1]:
        if item in prefs[person2]:
            si[item]=1

    # if they have no ratins in common, return 0
    if len(si)==0: return 0

    # Add up the squares of all the differences
    sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2) for item
        in si])

    return 1/(1+sqrt(sum_of_squares))









#method 1: distance metric
userSimilarityMatrix = calculateSimilarItems(prefs=prefs, similarityMetric=
    sim_distance, n=5, reverseSimilarityFlag=True, transformMatrixFlag=False) #
    reverseSimilarityFlag=True: largest to smallest
#userSimilarityMatrix = calculateSimilarItems(prefs=prefs, n=5,
    reverseSimilarityFlag=True, transformMatrixFlag=False, similarity=sim_pearson)

#print len(userSimilarityMatrix)
userSimilarityArrayOfTuples = []
for userId, userAttr in userSimilarityMatrix.items():

    totalSimilarity = 0
    similarUsersArray = []
    for scoreAnduserId in userAttr:
        score = scoreAnduserId[0]
        userId = scoreAnduserId[1]

        similarUsersArray.append(userId)
        totalSimilarity = totalSimilarity + score

```

Problem 8 continued on next page. . .

```
    userTupleData = (userId, similarUsersArray, totalSimilarity )
    userSimilarityArrayOfTuples.append(userTupleData)


count = 0
for userTuple in userSimilarityArrayOfTuples:

    userId = userTuple[0]
    userSimilarItems = userTuple[1]
    totalSim = userTuple[2]


    #any one of these qualifies
    #pick largest totalSim

    print userId, userSimilarItems, totalSim

print 'count: ', count
```

Figure 8: ques8.py at work



Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
vnwala@template:~$ python ques8.py
750 ['341', '814', '783', '925', '750'] 2.64626436994
613 ['341', '86', '857', '783', '613'] 2.02778586186
306 ['855', '845', '675', '631', '306'] 4.0
427 ['547', '273', '179', '111', '427'] 2.1086660909
696 ['93', '876', '762', '754', '696'] 5.0
89 ['908', '905', '903', '892', '89'] 5.0
598 ['866', '818', '845', '775', '598'] 3.5
713 ['856', '811', '772', '729', '713'] 2.5
762 ['939', '850', '848', '769', '762'] 5.0
845 ['923', '918', '900', '888', '845'] 5.0
715 ['850', '848', '793', '726', '715'] 5.0
792 ['943', '933', '903', '892', '792'] 5.0
688 ['912', '873', '820', '704', '688'] 5.0
855 ['920', '912', '860', '86', '855'] 5.0
848 ['935', '903', '892', '885', '848'] 5.0
744 ['911', '9', '88', '776', '744'] 5.0
859 ['928', '869', '863', '86', '859'] 5.0
685 ['898', '820', '818', '713', '685'] 5.0
799 ['911', '909', '876', '801', '799'] 5.0
712 ['670', '156', '929', '855', '712'] 3.5
35 ['863', '811', '570', '50', '35'] 5.0
750 ['898', '853', '816', '812', '750'] 5.0
51 ['925', '720', '700', '611', '51'] 5.0
```

Problem 9

9. What movie was rated highest on average by men over 40? By men under 40?
For men over 40 we have: 1) Great Day in Harlem, A (1994) 5.0 2) Two or Three Things I Know About Her (1966) 5.0 3) Aparajito (1956) 5.0 4) Strawberry and Chocolate (Fresa y chocolate) (1993) 5.0 5) Little Princess, The (1939) 5.0
For men under 40 we have: 1) Entertaining Angels: The Dorothy Day Story (1996) 5.0 2) Letter From Death Row, A (1998) 5.0 3) Hugo Pool (1997) 5.0 4) Leading Man, The (1996) 5.0 5) Quiet Room, The (1996) 5.0

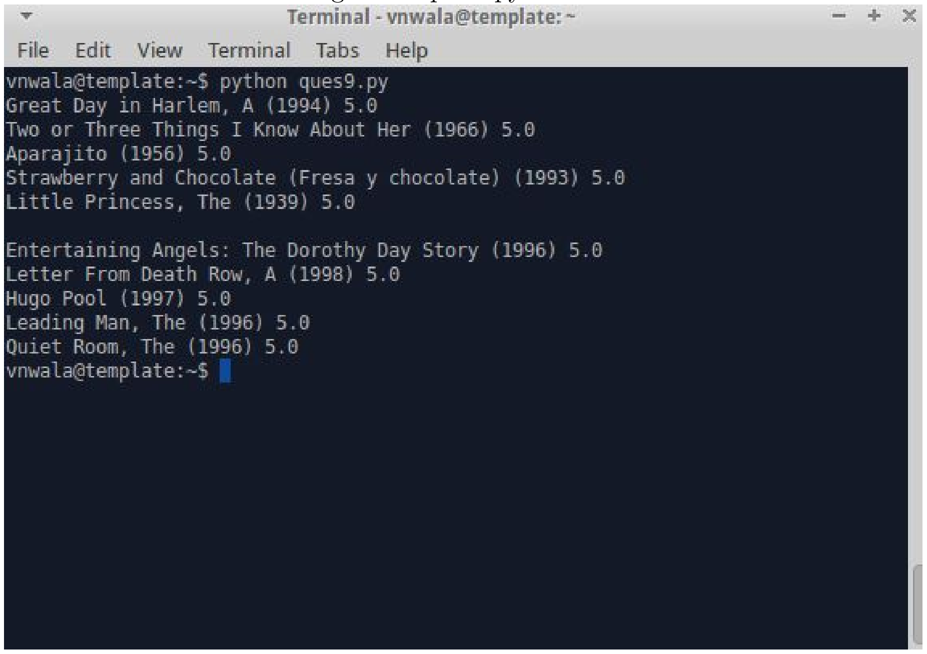
Listing 9: Python script solving problem 9

```python
def aggregateMovieAndUserData(path='/home/vnwala/ml-100k/'):

    try:

        movies = {}
        aggregateMovieData = []
        for line in open(path + 'u.item'):
            (id, title) = line.split('|')[0:2]
            movies[id] = (title, [], -1)

        users = {}
        #populate user and movie data

        for line in open(path + 'u.data'):
            (user, movieid, rating, ts) = line.split('\t')

            user = user.strip()
            movieid = movieid.strip()
            rating = rating.strip()
            ts = ts.strip()

            users.setdefault(user, {})
            #movie title: movies[movieid][0]
            users[user][movieid] = float(rating)#key is movie title

            '''
            if( user in users ):
                users[user][movies[movieid][0]] = float(rating)
            else:
                users[user] = {}
            '''


            #movies[movieid]: (title, [ArrayOfRatings])
            #movies[movieid][0]: title
            #movies[movieid][1]: array of ratings
            movies[movieid][1].append(float(rating))



        #process movie data
        for movieId, tupleData in movies.items():
            averageRating = sum(tupleData[1]) / float(len(tupleData[1]))
```

Problem 9 continued on next page. . .

```
            movietuples = (movieId, tupleData[0], averageRating, len(tupleData[1]))
            aggregateMovieData.append(movietuples)


        aggregateUserData = []
        for line in open(path + 'u.user'):
            (userId, age, gender, occupation, zipCode) = line.split('|')

            userTuples = (userId, gender, age, users[userId])
            aggregateUserData.append(userTuples)


    except:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(fname, exc_tb.tb_lineno, sys.exc_info() )
        return



    return aggregateMovieData, aggregateUserData, movies


def getFemaleAndMaleData(aggregateUsers):
    aggregateUsersFemale = []
    aggregateUsersMale = []
    if( len(aggregateUsers) > 0 ):

        for user in aggregateUsers:
            if( user[1] == 'F' ):
                aggregateUsersFemale.append(user)
            else:
                aggregateUsersMale.append(user)


    return aggregateUsersFemale, aggregateUsersMale




def getHighestRatedMoviesByMenOrWomenUnderAge(aggregateUsersFemaleOrMale, count, ageLimit, movies):

    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and ageLimit > 0 and len(movies) > 0 ):
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []
        for user in aggregateUsersFemaleOrMale:
            if( int(user[2]) < ageLimit ):

                for movie, rating in user[3].items():
                    if( movie in tupleOfMovieRatingDictionary ):
                        tupleOfMovieRatingDictionary[movie].append(rating)
                    else:
                        tupleOfMovieRatingDictionary[movie] = []
                        tupleOfMovieRatingDictionary[movie].append(rating)

        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)


        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)

        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[movieData[0]][0], movieData[1]

            if( i == count ):
                break
            i = i + 1

#input:
    #aggregateUsers: [ (user id, gender, Age, {'movie_title': movie rating}) ]







def getHighestRatedMoviesByMenOrWomenOverAge(aggregateUsersFemaleOrMale, count,
    ageLimit, movies):

    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and ageLimit > 0 and
         len(movies) > 0 ):
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []

        for user in aggregateUsersFemaleOrMale:

            if( int(user[2]) > ageLimit ):
                for movie, rating in user[3].items():

                    if( movie in tupleOfMovieRatingDictionary ):
                        tupleOfMovieRatingDictionary[movie].append(rating)
                    else:
                        tupleOfMovieRatingDictionary[movie] = []
                        tupleOfMovieRatingDictionary[movie].append(rating)
        #average ratings
        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)

        #sort
        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)

        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[movieData[0]][0], movieData[1]

            if( i == count ):
                break
            i = i + 1

aggregateMovies, aggregateUsers, movies = aggregateMovieAndUserData()

aggregateUsersFemale, aggregateUsersMale = getFemaleAndMaleData(aggregateUsers)

getHighestRatedMoviesByMenOrWomenOverAge(aggregateUsersMale, 5, 40, movies)
print ''
getHighestRatedMoviesByMenOrWomenUnderAge(aggregateUsersMale, 5, 40, movies)
```

Figure 9: ques9.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
vnwala@template:~$ python ques9.py
Great Day in Harlem, A (1994) 5.0
Two or Three Things I Know About Her (1966) 5.0
Aparajito (1956) 5.0
Strawberry and Chocolate (Fresa y chocolate) (1993) 5.0
Little Princess, The (1939) 5.0

Entertaining Angels: The Dorothy Day Story (1996) 5.0
Letter From Death Row, A (1998) 5.0
Hugo Pool (1997) 5.0
Leading Man, The (1996) 5.0
Quiet Room, The (1996) 5.0
vnwala@template:~$
```

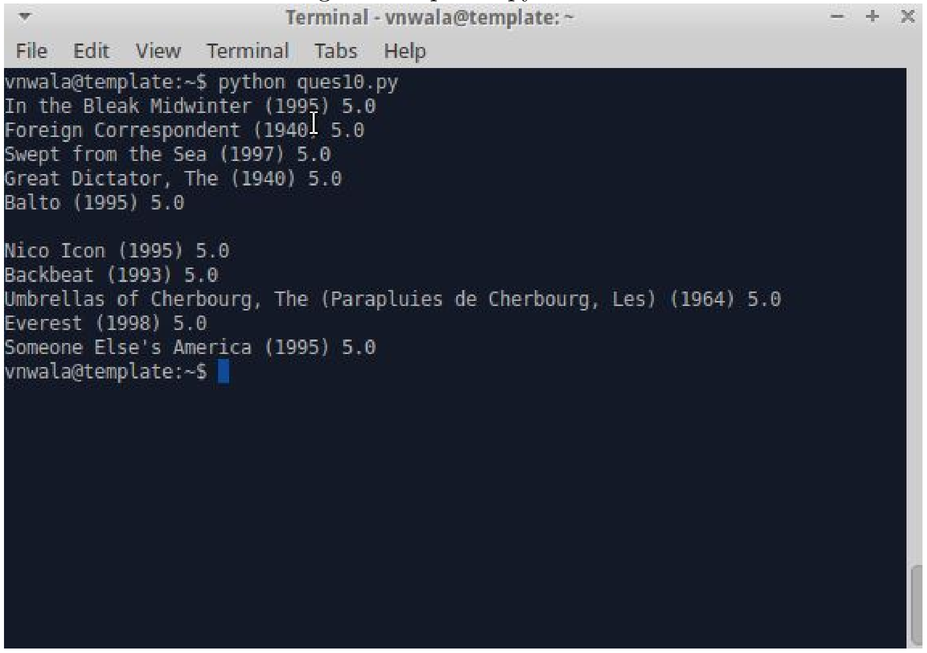
Problem 10

10. What movie was rated highest on average by women over 40? By women under 40?
For women over 40 we have:
1) In the Bleak Midwinter (1995) 5.0 2) Foreign Correspondent (1940) 5.0 3) Swept from the Sea (1997) 5.0 4) Great Dictator, The (1940) 5.0 5) Balto (1995) 5.0
For women under 40 we have: 1) Nico Icon (1995) 5.0 2) Backbeat (1993) 5.0 3) Umbrellas of Cherbourg, The (Parapluies de Cherbourg, Les) (1964) 5.0 4) Everest (1998) 5.0 5) Someone Else's America (1995) 5.0

Listing 10: Python script solving problem 10

```python
def aggregateMovieAndUserData(path='/home/vnwala/ml-100k/'):

    try:

        movies = {}
        aggregateMovieData = []
        for line in open(path + 'u.item'):
            (id, title) = line.split('|')[0:2]
            movies[id] = (title, [], -1)

        users = {}
        #populate user and movie data

        for line in open(path + 'u.data'):
            (user, movieid, rating, ts) = line.split('\t')

            user = user.strip()
            movieid = movieid.strip()
            rating = rating.strip()
            ts = ts.strip()

            users.setdefault(user, {})
            #movie title: movies[movieid][0]
            users[user][movieid] = float(rating)#key is movie title

            '''
            if( user in users ):
                users[user][movies[movieid][0]] = float(rating)
            else:
                users[user] = {}
            '''



            #movies[movieid]: (title, [ArrayOfRatings])
            #movies[movieid][0]: title
            #movies[movieid][1]: array of ratings
            movies[movieid][1].append(float(rating))




        #process movie data
        for movieId, tupleData in movies.items():
            averageRating = sum(tupleData[1]) / float(len(tupleData[1]))
            movietuples = (movieId, tupleData[0], averageRating, len(tupleData[1])
                )
            aggregateMovieData.append(movietuples)


        aggregateUserData = []
        for line in open(path + 'u.user'):
            (userId, age, gender, occupation, zipCode) = line.split('|')

            userTuples = (userId, gender, age, users[userId])
            aggregateUserData.append(userTuples)


    except:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
        print(fname, exc_tb.tb_lineno, sys.exc_info() )
        return



    return aggregateMovieData, aggregateUserData, movies


def getFemaleAndMaleData(aggregateUsers):
    aggregateUsersFemale = []
    aggregateUsersMale = []
    if( len(aggregateUsers) > 0 ):

        for user in aggregateUsers:
            if( user[1] == 'F' ):
                aggregateUsersFemale.append(user)
            else:
                aggregateUsersMale.append(user)


    return aggregateUsersFemale, aggregateUsersMale




def getHighestRatedMoviesByMenOrWomenUnderAge(aggregateUsersFemaleOrMale, count,
    ageLimit, movies):

    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and ageLimit > 0 and
         len(movies) > 0 ):
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []
        for user in aggregateUsersFemaleOrMale:
            if( int(user[2]) < ageLimit ):

                for movie, rating in user[3].items():
                    if( movie in tupleOfMovieRatingDictionary ):
                        tupleOfMovieRatingDictionary[movie].append(rating)
                    else:
                        tupleOfMovieRatingDictionary[movie] = []
                        tupleOfMovieRatingDictionary[movie].append(rating)

        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)


        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)

        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[movieData[0]][0], movieData[1]

            if( i == count ):
                break
            i = i + 1

#input:
    #aggregateUsers: [ (user id, gender, Age, {'movie_title': movie rating}) ]







def getHighestRatedMoviesByMenOrWomenOverAge(aggregateUsersFemaleOrMale, count,
    ageLimit, movies):

    if( count > 0 and count < len(aggregateUsersFemaleOrMale) and ageLimit > 0 and
         len(movies) > 0 ):
        tupleOfMovieRatingDictionary = {}
        movieAverageRatingArrayOfTuples = []

        for user in aggregateUsersFemaleOrMale:

            if( int(user[2]) > ageLimit ):
                for movie, rating in user[3].items():

                    if( movie in tupleOfMovieRatingDictionary ):
                        tupleOfMovieRatingDictionary[movie].append(rating)
                    else:
                        tupleOfMovieRatingDictionary[movie] = []
                        tupleOfMovieRatingDictionary[movie].append(rating)
        #average ratings
        for movie, ratingsArray in tupleOfMovieRatingDictionary.items():
            averageRating = sum(ratingsArray) / float(len(ratingsArray))

            movieRatingTuple = (movie, averageRating)
            movieAverageRatingArrayOfTuples.append(movieRatingTuple)

        #sort
        movieAverageRatingArrayOfTuples = sorted(movieAverageRatingArrayOfTuples,
            key=lambda tup: tup[1], reverse=True)

        i = 1
        for movieData in movieAverageRatingArrayOfTuples:
            print movies[movieData[0]][0], movieData[1]

            if( i == count ):
                break
            i = i + 1

aggregateMovies, aggregateUsers, movies = aggregateMovieAndUserData()

aggregateUsersFemale, aggregateUsersMale = getFemaleAndMaleData(aggregateUsers)

getHighestRatedMoviesByMenOrWomenOverAge(aggregateUsersFemale, 5, 40, movies)
print ''
getHighestRatedMoviesByMenOrWomenUnderAge(aggregateUsersFemale, 5, 40, movies)
```

Figure 10: ques10.py at work


Terminal - vnwala@template: ~
File Edit View Terminal Tabs Help

```
vnwala@template:~$ python ques10.py
In the Bleak Midwinter (1995) 5.0
Foreign Correspondent (1940) 5.0
Swept from the Sea (1997) 5.0
Great Dictator, The (1940) 5.0
Balto (1995) 5.0

Nico Icon (1995) 5.0
Backbeat (1993) 5.0
Umbrellas of Cherbourg, The (Parapluies de Cherbourg, Les) (1964) 5.0
Everest (1998) 5.0
Someone Else's America (1995) 5.0
vnwala@template:~$
```

Conclusion

To conclude, I should state that Alexander Nwala was a huge contributor for my answers from question 6 to 10, so those answers will in some or most cases have the same syntatic and functional properties similar to his code. Some questions were answered in part, depending on what I was able to do.

References

[1] arthur e. Programming-collective-intelligence. https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter2/recommendations.py, 24 Decenber 2012.

[]